European Security in Health Data Exchange

Deliverable D2.4

SHiELD SecDevOps – Initial version

<table>
<thead>
<tr>
<th>Editor(s):</th>
<th>Xabier Larrucea, Jason Mansell, Alberto Berreteaga</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible Partner:</td>
<td>TECNALIA</td>
</tr>
<tr>
<td>Status-Version:</td>
<td>final</td>
</tr>
<tr>
<td>Date:</td>
<td>30/08/2018</td>
</tr>
<tr>
<td>Distribution level (CO, PU):</td>
<td>CO</td>
</tr>
<tr>
<td><strong>Project Number:</strong></td>
<td>GA 727301</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>Project Title:</strong></td>
<td>SHIELD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Title of Deliverable:</strong></th>
<th>D2.4 – SHiELD SecDevOps – Initial version</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Due Date of Delivery to the EC:</strong></td>
<td>30/06/2018</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Workpackage responsible for the Deliverable:</strong></th>
<th>WP2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Editor(s):</strong></td>
<td>Alberto Berreteaga</td>
</tr>
<tr>
<td><strong>Contributor(s):</strong></td>
<td>All</td>
</tr>
<tr>
<td><strong>Reviewer(s):</strong></td>
<td>All</td>
</tr>
<tr>
<td><strong>Approved by:</strong></td>
<td>Alberto Berreteaga</td>
</tr>
<tr>
<td><strong>Recommended/mandatory readers:</strong></td>
<td>All WP’s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Abstract:</strong></th>
<th>This document represents the approach developed under SHiELD project for the SecDevOps approach.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keyword List:</strong></td>
<td>DevOps, Security</td>
</tr>
<tr>
<td><strong>Disclaimer</strong></td>
<td>This document reflects only the author’s views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein</td>
</tr>
</tbody>
</table>
## Document Description

### Document Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Modifications Introduced</th>
<th>Modification Reason</th>
<th>Modified by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>09/02/2017</td>
<td>Final version</td>
<td></td>
<td>Alberto Berreteaga (TECNALIA)</td>
</tr>
</tbody>
</table>
# Table of Contents

Table of Contents .................................................................................................................. 4
List of Figures .......................................................................................................................... 5
List of Tables ............................................................................................................................ 5
Terms and abbreviations ......................................................................................................... 6
Executive Summary ................................................................................................................... 7

1 Introduction .......................................................................................................................... 8
1.1 SHIELD DevOps development environment and procedures ............................................. 8
1.2 Version control and task management ............................................................................... 10
1.2.1 Software repository ...................................................................................................... 10
1.2.2 Tracking development ................................................................................................. 10
1.2.2.1 Version release ......................................................................................................... 11
1.3 Deployment management ................................................................................................. 11
1.3.1 Main functionality ........................................................................................................ 12
1.3.2 Integration points ......................................................................................................... 13
1.4 Infrastructure as specification ............................................................................................ 13
1.4.1 Main functionality ........................................................................................................ 13
1.4.2 Integration points ......................................................................................................... 14
1.5 Development conventions and best practices ..................................................................... 14

2 SHIELD DevOps integration and validation strategy ............................................................ 16
2.1 Integration stage: Strategy and process ............................................................................. 16
2.2 Requirements definition and validation: Strategy and process ......................................... 17
2.2.1 Requirements gathering and prioritization .................................................................. 17
2.2.2 Requirements documentation ....................................................................................... 17
2.2.3 Requirements continuous management ..................................................................... 18
2.2.4 Generic processes ....................................................................................................... 19
2.2.5 Requirements prioritization matrix ............................................................................ 20

3 Installation guidelines ........................................................................................................ 22
3.1 Setup configuration management for DevOps environment ................................................. 22
3.2 Get a (virtual) machine ..................................................................................................... 23
3.3 Enable access to the operator ........................................................................................... 24
3.4 Install the Docker engine ................................................................................................. 24
3.5 Create Jenkins Volume ..................................................................................................... 26
3.6 Install and execute the scripts for Jenkins and registry start-up ....................................... 26
3.7 Configuring Jenkins access keys ...................................................................................... 27
3.8 Configuring Jenkins task groups ...................................................................................... 27

Project Title: SHIELD Contract No. GA 727301

www.project-shield.eu
4 OpenNCP current development / deployment case ................................................................. 28
5 ShiELD’s OpenNCP development / deployment case ............................................................. 28
6 (Sec)DevOps development environment and procedures ....................................................... 28
7 Conclusions .......................................................................................................................... 28
8 References ............................................................................................................................. 28

ANNEX 1: SHiELD ecosystem generic processes and initial functionalities ................................ 30
  Process 1: Discovery of the public services ............................................................................. 30
  Process 2: Collect information from Non-Users ..................................................................... 30
  Process 3: Analyse the information (Big Data Analysis) ......................................................... 31
  Process 4: Generate recommendations for transforming public services ............................ 31
  Process 5: Generate KPI Reports .......................................................................................... 32
  Process 6: Provide Feedback from Users (Assessment) ......................................................... 32
  Process 7: Co-create Digital public services ......................................................................... 33

---

List of Figures

Figure 1. Stages for the development, integration and validation of the software components in SHiELD. ................................................................. 9
Figure 2. Tools for version controlling, deployment and infrastructure in SHiELD. ............... 10
Figure 3. SHIELD GitLab group and projects. ................................................................. ¡Error! Marcador no definido.
Figure 4. Jenkins environment in SHiELD ............................................................................ 12
Figure 5. Proposed integration strategy in SHiELD ............................................................... ¡Error! Marcador no definido.
Figure 6. Initial proposition of the requirements elicitation in SHiELD ecosystem ................................ ¡Error! Marcador no definido.
Figure 7. Requirements continuous management process ..................................................... 19
Figure 8. Colour code for requirements prioritization ............................................................ 20
Figure 9. SHIELD requirements prioritization matrix ............................................................ 21

---

List of Tables

Table 1. Requirements documentation template .................................................................... 18
Terms and abbreviations

<table>
<thead>
<tr>
<th>AB</th>
<th>Advisory Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>Consortium Agreement</td>
</tr>
</tbody>
</table>
Executive Summary

The objective of the present document is to reflect the approach for SHiELD SecDevOps which has been defined and developed within the SHiELD project. This document is a live document which is containing the latest modifications of the running prototypes.
1 SHIELD DevOps development environment and procedures

1.1 DevOps infrastructure set up

The SHIELD ecosystem will be composed of the provided ICT enablers to ease the implementation of the SHIELD approach. This SHIELD ecosystem will be formed by the different ICT components which need to be developed, assembled and executed.

The SHIELD software components will be developed by the technology partners of the consortium: XXXXXXXXXX. Each technology provider will set up its own development environment. Also, a shared software repository will be set up for the integration of the components so that they can interoperate, communicate and work together in an integrated manner. Finally, specific environments will be created for each Use Case, with the integration of the specific Key Results it utilizes.

In SHIELD, DevOps philosophy and the corresponding approach is used and applied at two levels:

- Internally for the development and the operation (deployment and validation into the use cases’ environments) of the software components forming the different ICT enablers in the ecosystem.
- Externally for the users of the SHIELD solution, as the different components will be integrated into the SHIELD DevOps framework for the development and operation of the SHIELD ecosystem.

The different ICT enablers which are part of the SHIELD ecosystem [2] are composed of several software components that will be implemented by different partners following different technologies. SHIELD will use a DevOps based approach able to fully support the management of these implementations and the planned releases. DevOps integrates development and operations into a single-minded entity [3] with common goals: high-quality software, faster releases and improved users’ satisfaction.

DevOps also incorporates a number of agile principles, methods, and practices such as continuous delivery, continuous integration, and collaboration [4].

The DevOps approach requires the set-up of a development and delivery pipeline that consists of the stages an application goes through from development through production, as exemplified in the following figure. It represents the environments that are envisioned in SHIELD covering the different development stages:
Figure 1. Stages for the development, integration and validation of the software components in SHIELD.

- The Development stage envisages the availability of a development environment that provides tools to write and test code, as well as to support collaborative development (e.g., source control management, work-item management, collaboration, unit testing, project planning). Possible tools and development approaches are: Git [5] as version control system, Apache Maven [6] to manage project's builds, reporting and documentation, and containerization technology to have applications running in self-contained units that can be moved across platforms (e.g., Docker [7]). TECNALIA will review and lead the process to ease the integration process as much as possible.

- The Integration stage focuses on compiling the code and performing the unit test and integration test reports. This stage also includes the availability of a common storage mechanism for the binaries created, as well as the assets required to deploy the applications (e.g., configuration files, infrastructure-as-code files, deployment scripts). Possible tools to support this stage are Jenkins [8] to support continuous integration and Apache Maven for building instructions.

- The Staging stage is where the QA, user acceptance, and development/testing teams do the actual testing. Possible tools to support this stage are again: Jenkins, Apache Maven for building and testing instructions, and xUnit for unit testing framework; as well as tools like Chef [9] (a cloud infrastructure framework that automates the building, deploying, and management of infrastructure); Docker or Puppet [10] (for data center orchestration by automating configuration and management of machines and software).

- The Production Environments envisage tools and features for application environment management and provisioning such as: Chef, Docker, Puppet, as well as tools like: Logstash [11] to speed up the feedback loop; osTicket [12], as ticket or issue tracking system.

In the following sections (2.2, 2.3, 2.4 and 2.5) the different tools shown in figure 2 and how they are planned to be used in the context of SHIELD are presented.
1.2 Version control and task management

1.2.1 Software repository

The technical work packages of SHIELD will use GitLab [13] to manage source code and for revision control. GitLab is a Web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git [5] as well as adding additional proprietary features.

The SHIELD GitLab will host both Private and Public repositories. The private repositories will be used to host the initial stages of the different components of the project until they are mature enough to be deployed into the public domain. Besides, the private repositories will be also used to store repositories required by the pilots to develop their pilot oriented specific source code and resources.

The different components of SHIELD will become public as defined in Open Source strategy and in the exploitation strategy [15]. Private repositories will host the partners' proprietary implementations as established in the different individual and collaborative dissemination strategies.

The Gitlab software repository for SHIELD is hosted in TECNALIA and has been organized as follows. SHIELD group has 5 projects, one per main component in the architecture.

The internal organization of the projects will be refined in the next phases of the project.

1.2.2 Tracking development

It is envisioned that SHIELD consortium will use a tool for managing the development project of the SHIELD platform comprising the different software components forming the SHIELD ecosystem. Several tools are being analysed for that purpose, such as Jira, or Gitlab issues.
Jira [16] is a tool for managing the development. A Jira Project is a collection of Issues, governed by schemes for Issue Types (bug, enhancement, feature request, etc.). It provides feature tracking, bug tracking, and project management functions.

A Jira setup includes also the following configuration parameters:

- **Issue Types**: Each issue has a specific type (Bug, Task, etc.). Each issue type has specific configurations such as workflow and screens.
- **Workflows**: A Workflow defines the transition of an issue through the various statuses during its lifecycle. A project has multiple workflows tailored to each issue type.
- **Screens**: A Screen determines which attributes are visible and when.
- **Permissions**: Allow to define who can do what to an issue.
- **Versions**: Issues can be assigned to versions. These collections of issues can then be tracked and managed as part of a roadmap.

The Issues management mechanism provided by Gitlab covers similar functionalities. The final selection of the development tracking will be made once the project is more advanced. This selection will become critical in the latest stages of the project during the maintenance and the pilot supporting phase. The initial stages are more subject to be managed by agile methodologies such as the backlog of scrum.

### 1.2.2.1 Version release

SHIELD project release schedule will be as follows:

- Major version release at each milestone as originally planned in the SHIELD Proposal [2] DoW.
- Minor version release following the alpha/Beta integration strategy proposed in section 3.1.

Versions are denoted using a standard triplet of integers: MAJOR.MINOR.INTERIM BUILD/PATCH (e.g. 1.2.51). In between Major or Minor version, partners can generate any number of interim releases.

### 1.3 Deployment management

In the context of SHIELD one of the main objectives of the DevOps philosophy is to enhance the flow between the development stage and the operation stage, in order to decrease the production times. There are several ways of improvement (e.g. Kaizen [17], Lean [18], Six Sigma [19]) but when the objective is to decrease the time of passing from development to production, one of the main resources is the systematic automation of repetitive tasks.

There exist multiple approaches for automating tasks. In SHIELD, it is envisioned to use Jenkins [8]. Jenkins is a quite extended automation server. Other alternatives for automating tasks are Bamboo [20], Travis [21], cruise-control [22], etc. These are the main reasons for selecting Jenkins:

- Open source license
- Support services available if necessary
- SHIELD partners previous experience
The usage of an automation server such as Jenkins, provides a lot of advantages when sharing the information about the status of the continuous integration tasks, both for the developers and for the users.

1.3.1 Main functionality

Jenkins server provides multiple functionalities that are extendable by adding plugins (more than 2000 available). In this section only those functionalities provided by Jenkins which are relevant for SHIELD will be described. We will focus only on some stages of the DevOps cycle like: registering the component, continuous integration, debugging, modifying the component and deleting it.

During the continuous integration stage, the automation server should support different development strategies, such as the alfa-beta approach for integration and manual testing (see section 2.1). The functionalities that will be needed for such a development strategy are:

- Creation of the automatization tasks to systematize DevOps cycle and accelerate the incorporation of the changes and modifications into the production environment.
- Grouping the tasks into different groups to manage complex development.
- Review the execution log so that we have mechanisms to identify the causes of a failure in the automation task.
- See the status of previous executions to analyze failures.
- Keep the results of past executions to see the detail of the past executions i.e. when a failure occurs.
- Notify the status of the executions for the automatic launching of some tasks.
- Recover the automation code from Gitlab, in order to include in configuration management the integration tasks.
- Delete projects.

Apart from these functionalities, using Maven or previously installed plugins, Jenkins will be able to manage the infrastructure to execute the different components so that it can perform the integration tests against them.

![Jenkins environment in SHIELD](image)

Figure 3. Jenkins environment in SHIELD
1.3.2 Integration points

With respect to the integration technologies, in SHIELD it is envisioned to use mainly the REST API provided by Jenkins. This API provides functionalities for:

- Recovering information from Jenkins
- Launching executions
- Creating/ Copying Jobs

1.4 Infrastructure as specification

In complex developments environments with technical incompatibilities or in order to assure the independency of the modules, the different components are allocated in different virtual (or physical) machines. This approach implies:

- Requesting and managing new virtual machines.
- Blocking resources that could be used for other projects.
- Configuring the access to those machines, including opening the corresponding ports etc.
- Configuring the operating system in each machine and the possible SSH access.
- Configuring the platform and the requirements of each of the components manually.

The containers technology allows the definition of separate spaces (both at communication level and at file system level) in the same virtual or physical machine, optimizing the computation resources.

At the same time, containers based technologies (such as Docker [7] or Warden) allows the explicit provision of the configuration of the containers: baseline operating system, packages included, initial content, etc. This allows the instantiation of the same container with exactly the same initial characteristics.

Furthermore, some containers technologies support the usage of the containers registry where developed containers can be uploaded so that other team members can download and use /test them with a small set of instructions.

In SHIELD it is envisioned to use Docker as containerization technology. These are the reasons for this election:

- Open source technology
- Professional support if required
- It provides a public registry for the containers or we can create our own one.
- It has an extensive and growing users’ community.

1.4.1 Main functionality

The Docker server provides a lot of functionalities. In this section only those relevant for SHIELD are going to be described. For this, we will focus on the following DevOps stages: continuous integration, publication, distribution and updating.

In SHIELD it is envisioned to use the following capacities from Docker:

- Definition of the platform requirements for the components.
- Containers creation including both the component and the platform requirements for it.
- Configuration of the containers during its instantiation.
• Logs communication
• Containers instantiation
• Containers instances stopping and deleting
• Persistence definition

With respect to communication it is envisioned that SHIELD project will use:

• Containers registration into the registry
• Project level registry creation

With respect to distribution it is envisioned that SHIELD project will use:

• Download containers from the registry

With respect to update it is envisioned that SHIELD project will use:

• Download containers versions from the registry

1.4.2 Integration points

With respect to the integration technologies, it is envisioned that SHIELD will use mainly the Maven plugin for Docker. This will allow us to obtain the actions registry (log) of what is happening in the different actions that supports and that are needed in the DevOps cycle:

• Build
• Run
• Stop
• Pushing into the registry
• Log

This registry integrates perfectly with Jenkins and allows us to analyze what has happened during the execution of the different activities.

The proposed approach may seem complex as it requires knowing the three technologies: Jenkins, Maven and Docker. But this approach provides the project with an editable configuration and adjustable to the needs of every project. All the files are stored with the project and are accessible and modifiable by the team working on it.

1.5 Development conventions and best practices

This section introduces the first set of naming conventions for the Software deliverables.

These software deliverables, which are actually a piece of code corresponding to one or part of a Key Result, will be accompanied by a short report explaining the main functionalities, technical design, downloading information, installation manual and licensing schema. The technical report will also have to follow the provided template for software deliverables.

Source files will follow the following naming convention:

eu.SHIELDh2020.modulename.componentname.subcomponentname.

Endpoints naming convention is as follows:

/SHIELD/[group]/[componentname]/
Domain names will be for the development environment: ***.dev.SHIELD-h2020.eu, while for the production environment this will be the name that it will be used: ****.SHIELD-h2020.eu

Source code files heading shall follow the following format:

/ *

* Copyright (c) 201x <<Company_name>>.
* All rights reserved. This program and the accompanying materials
* are made available under the terms of the
* <<licensing_schema_to_be_SHIELDed>> which accompanies
* this distribution, and is available at
* <<link of the information of the selected licensing schema>>
*
* Contributors:
*
* <<Full Name of the contributor(s)>> <<(Organization Name(s))>>
**Initially developed in the context of SHIELD EU project www.SHIELD-h2020.eu
*/
2 SHIELD DevOps integration and validation strategy

2.1 Integration stage: Strategy and process

As described in the previous section, the DevOps approach includes, among others, the integration stage. The integration stage includes the preparation of the different prototypes, compiling the source code, managing the relationships between the different components, performing the integration checks and their results. The integration stage implies that an available common storage for the binaries is needed, as well as for the assets required when deploying the application (i.e. configuration files, infrastructure-as-code files, deployment scripts). Tools that can support this integration stage in SHIELD are Jenkins and Maven.

The SHIELD integration stage will have two main environments:

- **α (Alpha) environment**: The alpha environment will serve to collect the source code of the different components (developed by different partner). The source code will be accompanied by their integration tests (developed by the main developers of the components). Both, the code collected for the prototypes and the code collected for the tests are built, run, and tested. When a component successfully passes these three phases it will be considered as a candidate to be included in the Beta environment.

- **β (Beta) environment**: The software components coming from the Alpha environment are tagged as Beta components, included in the Beta environment and a functional validation is performed by a set of experts. If the component has successfully passed the functional testing, it will become a candidate to be delivered as an official release of the prototype. This official release will be trespassed to the production environment where the corresponding use cases can validate it.

In the following picture, the graphical representation of the α and β environments is depicted:

The previous figure shows the different activities to be performed in the two environments:

- **α environment phase 1: Build** – This phase includes the creation of the Jenkins tasks (functionality and data) for getting the source code from the software repository (gitlab) and the Docker image will be created. This phase may also include the creation of the Jenkins tasks for getting specific pre-requisites needed to build the image (i.e dependencies to specific libraries or needed endpoints). As a result of this phase, a docker image will be created for the software component and its corresponding integration tests (which have been developed by the components development team). The management of the dependencies needed to compile the components will be in charge of the Maven client included in the Jenkins framework.

- **α environment phase 2: Run** – During this phase the Jenkins tasks for running the component and setting up the alpha environment are created and performed.

- **α environment phase 3: Test** – During this phase the Jenkins tasks for running the integrations tests need to be created and performed. A layers based approach will be followed for the definition of the tests, from the simplest (without dependencies) to the most complex ones, with dependencies upon the previous tests:
  - 0.0 Tests for loading data
  - 0.1 Tests of functions which need data to be executed
  - 0.2 Functions which need 0.1 functions to be executed
  - Etc.
When the integration tests are successfully passed the Beta phase starts with the following activities:

- Tag the components as Beta components.
- Set up the Beta environment
- Functional testing by a group of experts.

If the software component successfully achieves and passes the final step, it will become an official SHIELD software release, to be validated by the SHIELD use cases. If the final tests are not successfully passed, the component will come back to the alpha environment.

### 2.2 Requirements definition and validation: Strategy and process

This section introduces how the different functional requirements of the different components in the SHIELD ecosystem have been elicited and prioritized.

In the current document, the process for the elicitation of the requirements is detailed. The proposed process is iterative and comprises the following steps:

1. Requirements gathering
2. Requirements prioritization
3. Requirements documentation
4. Continuous requirements management

#### 2.2.1 Requirements gathering and prioritization

The prioritization and gathering of SHIELD requirements is iterative. During the first months of the project the followed process has been:

1. Definition of main generic processes and the main functionalities of the SHIELD ecosystem.
2. Considering these generic processes, they were decomposed into functional and non-functional requirements.
3. Identification of the requirements coming from the SHIELD scenarios.
4. Alignment of the requirements coming from both sources.
5. Prioritization of the requirements considering:
   a. Generic functionalities which are needed for the correct functioning of the SHIELD ICT based ecosystem
   b. Use cases needs to be able to validate the different versions of the SHIELD ecosystem prototype. Both the technical providers and the use cases will participate in this activity.
6. Requirements review considering the validation performed by the use cases.
7. Prioritize the requirements for the next validation.
8. Repetition of the steps 6 and 7.

#### 2.2.2 Requirements documentation

Once the requirements have been gathered and prioritized, they will be documented following the template presented next and will be assigned to each responsible partner for their implementation. The requirements will be added to the project plan. The reporting of the requirements implemented in each of the iterations and the planning for the implementation of the requirements in next iterations will be reported as part of the documentation to be delivered with the technical design of each of the different components.
### Table 1. Requirements documentation template

<table>
<thead>
<tr>
<th>Requirement id</th>
<th>Req. short title</th>
<th>Req. description</th>
<th>Type</th>
<th>Source</th>
<th>State</th>
<th>Priority</th>
<th>Reference documents</th>
<th>Additional comments</th>
<th>Relate KR</th>
</tr>
</thead>
</table>

2.2.3 Requirements continuous management

The process is shown in the next picture:

---

1 For the requirements prioritization SHIELD will use the MoSCoW method ([http://www.coleyconsulting.co.uk/moscow.htm](http://www.coleyconsulting.co.uk/moscow.htm)): Must have, Should have, Could have, Won’t have. The requirements will be prioritized based on the needs of the SHIELD use cases tool.
In each of these phases in the requirements management process, their state will change based in the situation they are. The proposed states are described next:

- **Proposed**: A project partner (social or technical expert) has provided a requirement that needs to be evaluated. Evaluation criteria include, the use cases need, the technical complexity, the existence or not of similar tools, alignment with the general objective or with the individual objectives of the project, coverage a KR, etc.
- **Accepted**: The requirement has been accepted for its implementation.
- **Rejected**: The requirement has not been accepted.
- **Implemented**: The requirement has been implemented, but it has not been implemented by any use case.
- **Work in progress**: The requirement is being implemented.
- **Finalized**: The requirement has been implemented and validated at least in one use case.

### 2.2.4 Generic processes

The requirements will be validated in two ways. On one hand, the correct implementation of the related functionality and its correct functioning with a fictitious, but representative, dataset will be validated. This validation is included in the implementation phase. On the other hand the validation of the requirements will be performed in almost real environments, that is, in the SHIELD use cases.

Next, the generic functionalities that the SHIELD ecosystem should provide are listed. In the current iteration, the functionalities required by the use cases are not covered. These are the basic and ideal functionalities that the ecosystem should provide. These generic processes will be presented to the consortium to assess that they cover they expectations.

- **Process 1 - Discovery of public services**:
- **Process 2 - Collect information from non-users**
- **Process 3 – Analyze the information**
- **Process 4 – Generate recommendations for transforming public services**
- **Process 5 – Generate KPI reports**
- **Process 6 – Provide Feedback from Users (Assessment)**
- **Process 7 - Co-create Digital public services**
These processes have been used to detail the initial set of functionalities which at the same time have been used to detail the first version of the SHIELD ecosystem architecture. The generic functionalities that were defined are:

- Security management
- Recommendation generation
- Digital maturity assessment
- Digital Public Services Management
- Legislation management
- Data analysis
- KPI report generation
- SHIELD management console
- Co-creation
- Data Bases
- UI
- Business model

The processes are detailed in Annex 1.

2.2.5 Requirements prioritization matrix

For the prioritization of the requirements in SHIELD several aspects have been considered:

- Baseline functionalities will be implemented in the initial versions, while more complex functionalities depending on the basic ones will be implemented in subsequent releases.
- Functionalities which are core for the use cases have been prioritized for the initial releases, based on the use cases description and brainstorming sessions with the use cases. This prioritization of the functionalities will be accordingly updated as the description and needs of the use cases evolve.
- Functionalities with strong dependencies with other tools (i.e. interfaces related functionalities) will be implemented in early stages of the project, so that the integrated framework can be built up and the components can test the dependencies with other components.

The initial prioritization of the SHIELD functional requirements is shown in the next table. The colours code is as follows:

![Figure 5. Colour code for requirements prioritization](image)

Project Title: SHIELD

Contract No. GA 727301

www.project-shield.eu
<table>
<thead>
<tr>
<th>Component and functional requirements</th>
<th>Functional requirement implementation plan</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intelligent Discovery (ID)</strong></td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td>ID.01</td>
<td>x</td>
</tr>
<tr>
<td>ID.02</td>
<td>x</td>
</tr>
<tr>
<td>ID.03</td>
<td>x</td>
</tr>
<tr>
<td>ID.04</td>
<td>x</td>
</tr>
<tr>
<td><strong>Digital Maturity Assessment (DigiMat)</strong></td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td>Co-creation Methodology supporting tool (CC)</td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td><strong>KPI visualization and report generation (KP)</strong></td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td>KP.01</td>
<td>x</td>
</tr>
<tr>
<td>KP.02</td>
<td>x</td>
</tr>
<tr>
<td>KP.03</td>
<td>x</td>
</tr>
<tr>
<td>KP.04</td>
<td>x</td>
</tr>
<tr>
<td><strong>User Assessment (UA)</strong></td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td>UA.01</td>
<td>x</td>
</tr>
<tr>
<td>UA.02</td>
<td>x</td>
</tr>
<tr>
<td>UA.03</td>
<td>x</td>
</tr>
<tr>
<td>UA.04</td>
<td>x</td>
</tr>
<tr>
<td><strong>Security Management</strong></td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td><strong>Anonymization (AN)</strong></td>
<td>M15 M30 M33</td>
</tr>
<tr>
<td>AN.01</td>
<td>x</td>
</tr>
<tr>
<td>AN.02</td>
<td>x</td>
</tr>
</tbody>
</table>

Figure 6. SHIELD requirements prioritization matrix.
3 Installation guidelines

In order to achieve in SHIELD the development, integration and validation strategies described in previous sections, the DevOps environment has been setup. As explained, it allows version control, continuous integration and automated deployment management.

In this chapter, we describe how the different components of the DevOps environment are organized and provide the main guidelines for installing them.

To deploy the environment, the following steps have been followed:

0. Setup a machine for the integration and deployment part
1. Enable operator access
2. Install the Docker engine
3. Create the volume of Jenkins
4. Install the Jenkins startup scripts and log
5. Start the Jenkins startup scripts and log
6. [Configure the Jenkins access keys] Ponerlo???
7. Configure the Jenkins task groups

3.1 Setup configuration management for DevOps environment

The DevOps approach in complex projects such as SHIELD requires large infrastructures with custom setup of network, computing and storage resources. Besides, the project will run for several years, years in which many changes may come including people changes and organisational changes. And more over, afterwards years after the project ends it could be necessary to restart the infrastructure to support exploitation opportunities or even contractual requests.
Therefore it is critical to preserve the environment for long time. This can be done in two ways, keeping the environment for as many years as necessary or gathering the knowledge and the resources required to re-establish it. Keeping the environment involves high operational costs, starting from thousands of euro each year, while keeping the knowledge do not.

In SHiELD we will support the second approach, we will keep the knowledge to be able to establish again the infrastructure when needed. This is known as IaaS (Infrastructure as code). IaaS is a research topic, were there is not a consensus on the procedures and technologies required to do it. There are some pieces, tools and approaches, which may cover some part of what IaaS may imply. In SHiELD we will some of them such as docker or Jenkins. For those parts of the infrastructure definition where we have no clear approach to formalise the IaaS approach we will use scripts and traditional documentation.

These entire resources docker, Jenkins, scripts and documentation should be managed as code, and for that reason we will setup a git repository to gather all the elements, excluding the binary packages, required to rebuild the integration environment from scratch. Binary packages such as Operating systems, binary packages, container images, java dependencies,... will be gathered from official repos using standard procedures such as: vagrant box add, apt-get install, yum install, docker pull, maven dependency:get, ...

### 3.2 Get a (virtual) machine

In this case the machine is provided by the Data Centre of Tecnalia. We have installed Ubuntu 16.04 on it.

- **IP address:** 172.26.252.126
We defined the integration machine under a conservative approach as it is fairly easy to increase the virtual machine resources at any moment.

### 3.3 Enable access to the operator

The machine has an administrator account, with *sudo* permissions provided -to avoid the need of provide the keys continuously-. The operator in this case is a Windows user, and we use Putty to access the Linux machine from Windows.

The access to the account is based on a certificate, so you need to:

- Create a new certificate: we have used the Puttygen tool to generate it.
- Configure the Linux account for the certificate: add a line containing the generated public key to the `.ssh/authorized_keys` file
- Configure in Putty the SSH access account to use the credential: in the options SSH/Auth we choose the private key previously stored.

![PuTTY Configuration](image.png)

### 3.4 Install the Docker engine

Listar el script para instalarlo?

to install docker server

```
ssh esilab@integration-operando.westeurope.cloudapp.azure.com

sudo su

apt-get update
```
apt-get -y upgrade

apt-get install -y apt-transport-https ca-certificates

apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBFB76221572C52609D

if your are prompted error like this

```
[gnupg: Invalid option "--recv-keys"]
```

apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv 58118E89F3A912897C070ADBFB76221572C52609D

Let’s continue

echo "deb https://apt.dockerproject.org/repo ubuntu-xenial main" | sudo tee /etc/apt/sources.list.d/docker.list

apt-get update

for uafs storage driver (not sure if really needed but in case of)

apt-get install -y linux-image-extra-$(uname -r) linux-image-extra-virtual

apt-get install -y docker-engine

service docker start

Another important thing to do is to enable the docker sock to any user after each reboot, I know that this is dangerous but is the only way I now to make the host docker receive orders from the Jenkins docker

```bash
sed -i -e "s/exit 0/chmod 666 /var/run/docker.sock
exit 0/" /etc/rc.local
```

Create the data folder

```bash
mkdir /data
```

We log into the azure virtual machine.

As part of the installation we also move the docker folder inside the data folder, to do so we will create a symlink.

```bash
service docker stop
mv /var/lib/docker/ /data/
ln -s /data/docker /var/lib
ll /var/lib/docker/
```

service docker start
3.5 Create Jenkins Volume

Listar el script para crearlo?

One created, you can list the volumes, and optionally remove the existing one to start from a clean one.

```
esilab@cpd01:~$ docker volume list
DRIVER       VOLUME NAME
Local         jenkins_home
esilab@cpd01:~$ docker volume rm jenkins_home
esilab@cpd01:~$ docker volume create --name jenkins_home
```

3.6 Install and execute the scripts for Jenkins and registry start-up

Start-up scripts have to be created. The first one (Jenkins.sh) starts the Jenkins and gives you access to Docker socket, so you can create Docker images and boot them. The second one (registry.sh)

These are the scripts we have defined:

`/data/jenkins.sh`

```
#docker run -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home jenkins &
chmod 666 /var/run/docker.sock
docker run --restart=always -p 8080:8080 -p 50000:50000 --name=eu.SHiELD.jenkins -v jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock jenkins &
```

`/data/registry.sh`

```
docker run -d -p 5000:5000 --restart=always --name eu.SHiELD.registry registry:2
```

When executed, the first script creates an admin user and generates a password. Using them, we perform an initial configuration.
The second script just starts the registration.

### 3.7 Configuring Jenkins access keys

No me parece que este paso tenga entidad suficiente para ponerlo...

Besides the administration account we will create two additional accounts in order to facilitate the interaction of the developers with the integration environment:

- Observer
- Auto

The observer account purpose will be to enable the developers to access the compilation results of their components, that way they can investigate possible reasons of problems of compilation in the integration environment.

The Auto account purpose is to enable the triggering of Jenkins jobs through a REST request.

```
curl -X POST http://jenkins.devops.SHIELD.esilab.org:8080/job/eu.SHIELD.int.digimat.server.jnk.tsk/buildWithParameters --user auto:password --data token=secrettoken
```

### 3.8 Configuring Jenkins task groups

Finally, you have to define the task groups. The task groups we have configured for SHIELD are the following ones:

- 00-Automation
- 01-Prerequisites
- 02-Images
- 03-Services
- 04-Test
4 OpenNCP current development / deployment case

5 SHiELD’s OpenNCP development / deployment case

6 (Sec)DevOps development environment and procedures

7 Conclusions

This deliverable presents the approach taken in the SHIELD project to set up the SHIELD DevOps Infrastructure, including a description on how DevOps philosophy will be implemented in the context of the development and operation of the ICT enablers in SHIELD.

Apart from the strategy followed and the tools used, the document introduces the integration and validation strategy followed in the context of the SHIELD ecosystem. Finally it also includes the installation guidelines for the DevOps infrastructure.

The set up environment along with the integration and validation strategy will be the basis for the development, integration and operation of the different software components to be implemented as part of the SHIELD ecosystem.
ANNEX 1: SHIELD ecosystem generic processes and initial functionalities

Process 1: Discovery of the public services

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process id</td>
<td>Discover public services</td>
</tr>
<tr>
<td>Objective</td>
<td>To discover the most suitable service for concrete citizens.</td>
</tr>
<tr>
<td>Description</td>
<td>1. Citizens request discovering/updating Public digital services.</td>
</tr>
<tr>
<td></td>
<td>2. Citizens may specify some text to drive the search</td>
</tr>
<tr>
<td></td>
<td>3. SHIELD Platform consults the citizens’ profile.</td>
</tr>
<tr>
<td></td>
<td>4. Based on the profile, the platform searches the most suitable public</td>
</tr>
<tr>
<td></td>
<td>services from the ones offered by the PA</td>
</tr>
<tr>
<td></td>
<td>5. SHIELD Platform presents the Citizen those services available for him.</td>
</tr>
<tr>
<td>Actors</td>
<td>Citizens (users)</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Catalogue of public digital services duly populated</td>
</tr>
</tbody>
</table>

Process 2: Collect information from Non-Users

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process id</td>
<td>Collect information from Non-Users</td>
</tr>
<tr>
<td>Objective</td>
<td>The objective if this process is to collect the relevant information</td>
</tr>
<tr>
<td></td>
<td>obtained from the citizens who are not using digital public services. This</td>
</tr>
<tr>
<td></td>
<td>information will be gathered from the Open Data portals.</td>
</tr>
<tr>
<td>Description</td>
<td>1. Identify who are the non-users of the services. (This information can</td>
</tr>
<tr>
<td></td>
<td>be obtained from the analysis of the people visiting the Citizens Visitor</td>
</tr>
<tr>
<td></td>
<td>Centres analysed in WP2).</td>
</tr>
<tr>
<td></td>
<td>2. Establish which information we need from non-users (information provided</td>
</tr>
<tr>
<td></td>
<td>by WP2)</td>
</tr>
<tr>
<td></td>
<td>3. Send the relevant information to be analyzed in the data analysis</td>
</tr>
<tr>
<td></td>
<td>process.</td>
</tr>
<tr>
<td>Actors</td>
<td>Triggered by other processes (or by the SHIELD operator, i.e. a PA servant)</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Open Data portals identified</td>
</tr>
<tr>
<td></td>
<td>Criteria for non-user identification well defined</td>
</tr>
<tr>
<td>Data Sources</td>
<td>Open Data portals</td>
</tr>
</tbody>
</table>

Contract No. GA 727301
www.project-shield.eu
### Process 3: Analyse the information (Big Data Analysis)

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Use Case id</td>
<td>Analisys of information</td>
</tr>
<tr>
<td>Objective</td>
<td>The objective of this process is to analyse the available relevant information to provide this information to the appropriate other processes</td>
</tr>
</tbody>
</table>
| Description      | 1. Load the context provided by PA or Citizens.  
2. Check which information is relevant for that context (**this information should be provided by the WP2 / WP3**) For example, If you want to analyse the digital maturity of the PA, the number of digital services is relevant, or if you want to improve your Birth digital services, the number of women between 18 and 40 is relevant….  
3. Search all the related data based on step 2. This search will be done using open data portals (for example launching the process ‘**collect information from Non-Users**’), but not only. Depending on the context, this search can be performed using other data sources like for example: citizens’ profiles, PA databases, feedback databases and so on.  
4. Carry out a Big data analysis to generate statistically aggregated or otherwise processed data.  
5. Store the generated data (and keep the queries).  |
| Actors           | Triggered by other processes                                                                                                                                                                                |
| Pre-Conditions   | This process needs to have which information is relevant to be analysed in each context: co-creation, personalisation of services and so on                                                                    |
| Data sources     | Open data portals  
Use case data bases  
Feedback from citizens (Assessment service and social networks)                                                                                                                                         |

### Process 4: Generate recommendations for transforming public services

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Use Case id</td>
<td>Generate recommendations.</td>
</tr>
<tr>
<td>Objective</td>
<td>The objective of this process is to elaborate a set of contextualized recommendations and guidelines for transforming Public policies and processes.</td>
</tr>
</tbody>
</table>
Description

1. The PA identifies the context for the recommendations (‘What does the PA want to do?)? For example: Improve the digital maturity, improve the usage a concrete services (Birth Services), etc.
2. Launch the process ‘**Analyse the information**’.
3. Using the relevant data generated and applying the rules created in **WP2**, generate the report with the appropriate recommendations for the specific context.

<table>
<thead>
<tr>
<th>Actors</th>
<th>PAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Conditions</td>
<td>Generic Rules for the recommendations provided by WP2</td>
</tr>
<tr>
<td>Data sources</td>
<td>Open data portals</td>
</tr>
<tr>
<td></td>
<td>Use case data bases</td>
</tr>
<tr>
<td></td>
<td>Feedback</td>
</tr>
</tbody>
</table>

**Process 5: Generate KPI Reports**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Use Case id</td>
<td>Generate KPI reports</td>
</tr>
<tr>
<td>Objective</td>
<td>The objective of this process is to provide to the PAs with KPI reports to help them to improve their policies and processes.</td>
</tr>
</tbody>
</table>
| Description                   | 1. PA identifies the context for the KPIs report, for example which services are the most used ones, which is the usage history, how many citizens have profile in the system, etc. **The relation between the KPIs and the context will be provided by WP3, WP2 and WP5**  
2. Launch the process ‘**Analyse the information**’  
3. With big data analytics techniques, derive those KPIs and their trends that will help the PAs to take the correct decisions  
4. Present to the PAs a graphical representation of the the KPIs and their trends |
| Actors                        | PAs                                                                        |

**Process 6: Provide Feedback from Users (Assessment)**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Use Case id</td>
<td>Provide feedback from users (citizens and governments)</td>
</tr>
<tr>
<td>Objective</td>
<td>To allow users to provide feedback from a certain digital service.</td>
</tr>
</tbody>
</table>
### Description

1. Citizen selects the service in order to provide feedback
2. The SHIELD Assessment service provides the questions to assess the service
3. Citizens answers these questions with his feedback
4. The SHIELD Assessment service stores the feedback to be analysed by the process ‘**Analyse the information**’.

### Actors

Citizens

---

### Process 7: Co-create Digital public services.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Use Case id</td>
<td>Co-Create</td>
</tr>
<tr>
<td>Objective</td>
<td>To define / change a digital public services in a collaborative manner between PAs and Citizens</td>
</tr>
</tbody>
</table>
| Description   | 1. PA involves relevant citizens for co-creation. (The kind of relevant citizens – **provided by WP3** - could be provided by the KPIs report, by the context, by the experience of the PA, etc.)  
2. Customise the co-creation methodology (**WP3**) based on the nature of the legal system, nature of the service to be co-created, the target groups and so on.  
3. Co-create the service |
| Actors        | Citizens (users and non-users) PAs                                          |